

Tervezési minták egy OO programozási nyelvben (Software Design Patterns)

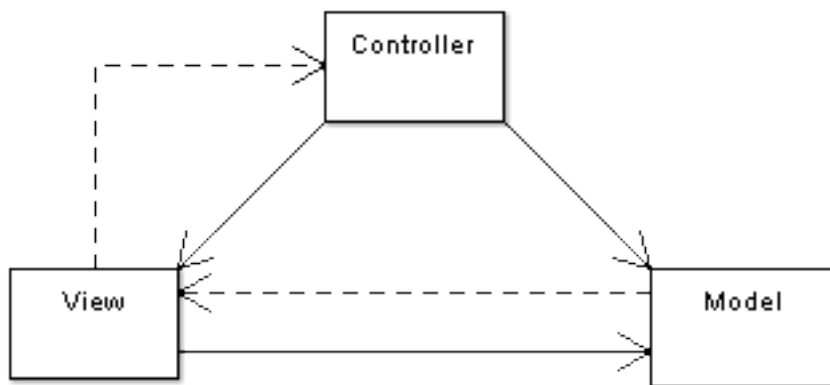
Definíció Egy programtervezési minta egy programozási feladatra ad általános, újrafelhasználható megoldást. Egymással együttműködő objektumok és osztályok leírása.

1. MVC modell

Az MVC modell jelentése: Modell - View - Controller. A felhasználó a nézet réteggel lép interakcióba, műveletet végez. Ezt a műveletet, utasítást feldolgozza a controller réteg. Az információkat, a program számára kezelhető formában, objektumokban (ValueObject) a modell réteg kezeli.

Kötelező rétegek a fentebb leírtak szerint: - **Modell** : objektumok leírása. Kizárólag adatot tartalmaz, gettert és settert ha ez lehetséges. Sokszor az adattagok nem módosíthatóak, egyszer kaphatnak értéket a konstruktor hívásakor. - **Controller** : Felhasználói műveletek kezelése. A műveletek elvégzéséhez szükséges adatokat a modell objektumokban tárolja, ehhez hozzáveszi a felhasználtól kapott paramétereket. - **View** : felhasználói felület megjelenítése, utasítások vétele.

Lehetnek opcionális rétegek is: - **Service** : egy közvetítő réteg információt kér a modelltől és továbbítja a vezérlőnek. - **Persister** : tárolja valamilyen perzisztens módon (adatbázis, file), azokat az adatokat amiket később, a program esetleges újraindítása után még használni kell.



MVC modell

2. Néhány további tervezési minta:

1. Objektum létrehozására vonatkozó tervezési minták:
 - **Prototype** : definiálunk és létrehozunk egy objektumot, majd másolatot készítünk belőle ha új példányra van szükség.
 - **Singleton** : létrehozunk egy osztály egy példányát, kizárólag egy helyen.
Majd globális hozzáférést biztosítunk azon elemeknek, akik ezt használják.
Pl.: adatbázis elérés
2. Osztályok felépítésére vonatkozó minták:
 - **Decorator** : Az objektumok wrapper osztályokba csomagolhatóak, amelyek egy viselkedést valósítanak meg.
Pl.: GUI esetében egy felirat egy gomb nevű wrapper osztályba csomagolva már kattintható, egyébként nem.
 - **Bridge** : Egy osztályt két részre osztunk : absztrakcióra és implementációra. Majd egymástól függetlenül fejlesztjük őket.
3. Objektumok közötti kapcsolatra vonatkozó tervezési minták:
 - **Mediator** : Összetett függőségek egyszerűsítésére jó. Az objektumok közvetlenül nem kommunikálnak a függőségeikkel. Egy közvetítő objektumnak (mediator) küldik az üzenetet, ez a függőségek segítségével megoldja a feladatot, majd az eredményt visszaküldi a kérés feladójának.
 - **Command** : Minden utasítást egyetlen objektumban írunk le úgy, hogy ez tartalmazzon minden információt a végrehajtáshoz. Felkészítjük a végrehajthatatlan utasításra is. Hasznos ha az alkalmazás sok utasítást tud kezelni, az MVC-vel összevetve a View rétegben a felhasználói utasítások egységes kezelését teszi lehetővé, ugyanakkor belső utasításokat is könnyebb vele kezelni.
 - **Visitor** : Szétválasztja a metódusoktól az objektumokat, amelyeken műveletet végez. Megfigyelhető a hasonlóság az MVC-modellbeli model-service szervezésben is, a modell osztály itt sem tartalmaz semmiféle metódust a getteren és az esetleges setteren kívül, mindent a service intéz. Használható lehet ez a szoftver fő működésének elrejtésére is, egy esetleges lehallgatásból csak az adatszerekezt derül ki a mechanika nem.
 - **Template method** : Általános viselkedés leírását teszi lehetővé. Egy osztály

definál egy algoritmust metódusokon keresztül, majd az egyes leszármazottak felülírhatják ezeket. Konkrét megvalósításai ennek az interface, abstract metódusokkal.

Források

Design patterns - Wikipedia
MVC - Wikipedia
MVC schema - ELTE Inf
Design Patterns Catalog